Energy-Efficient Packet Processing

Ravi Kokku Upendra B. Shevade Nishit S. Shah Mike Dahlin Harrick M. Vin
email: {*rkoku, upendra, nishit, dahlin, vin*}*@cs.utexas.edu*
University of Texas at Austin Technical Report # TR04-04.
Updated version of the report: *http://www.cs.utexas.edu/users/rkoku/RESEARCH/energy-tech.pdf*

Abstract

Packet processing systems (e.g., routers, virus scanners, intrusion detectors, SSL accelerators, etc.) provision sufficient number of processors to handle the expected maximum workload. The observed load at any instant, however, is often substantially lower; further, the load fluctuates significantly over time. These properties offer an opportunity to conserve energy (e.g., by deactivating idle processors or running them in low-power mode). In this paper, we present an on-line algorithm for adapting the number of activated processors such that (1) the total energy consumption of the system across a packet trace is minimized and (2) the additional delay suffered by packets as a result of adaptation is deterministically bounded. The resulting Power Management Algorithm (PMA) is simple, but it accounts for system reconfiguration overheads, copes with the unpredictability of packet arrival patterns, and consumes nearly the same energy as an optimal off-line strategy. A conservative version of the algorithm (CPMA), turns off processors less aggressively than is optimal but still provides good energy savings while reducing the additional packet latency introduced by power management. We demonstrate that for a set of trace workloads both algorithms can reduce the core power consumption by 60-70% while increasing the average packet processing delay by less than 560µs (PMA) and less than 170µs (CPMA).

1 Introduction

Design of energy-efficient networks is an emerging area of research. In this paper, we focus on the specific subproblem of designing energy-efficient *packet processing systems (PPS)*. We demonstrate that our algorithm minimizes the total energy consumption of each packet processing system while ensuring that the additional delay suffered by packets as a result of adaptation is bounded. In what follows, first we discuss the problem, the opportunity, and the challenges in designing energy-efficient packet processing systems. Then, we outline the contributions of this research.

The Problem The Internet infrastructure today includes a wide-range of packet processing systems (e.g., routing, firewall, VPN, IPv4/v6 gateway, intrusion detection, virus scan, and load balancing) whose cumulative energy demands are substantial [1, 5, 14, 15]. For instance, Gupta et al. [14] show that the hubs, switches, and routers deployed in the Internet during the year 2000 in the United States consumed about 6 TW-h of energy costing about one billion dollars per year[1] in operating expense or roughly the output of one nuclear reactor unit; to deploy and operate a similar Internet infrastructure in the rest of the world would require over 100 TW-h of electricity.

The cumulative energy needs of the Internet infrastructure is expected to increase with (1) the growth of the Internet (especially in developing countries); (2) the increases in the diversity and complexity of applications supported by PPS; and (3) the increases in the performance levels supported by PPS. This trend is evident from the increase in the power requirements of different generations of network processors: for example, Figure 1 summarizes the increasing power requirements of increasingly capable members of the Intel's IXP family of network processors.

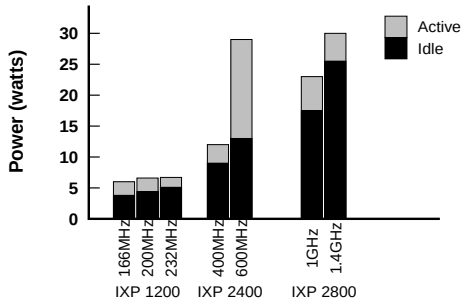

Figure 1: Power requirements of Intel's IXP family of processors.

The Opportunity Two trends facilitate the design of energy-efficient packet processing systems. First, to achieve robustness to traffic fluctuations, PPSs are often provisioned with sufficient processing resources to handle the expected *maximum* traffic load. However, as illustrated by Figure 2, the observed load fluctuates significantly over time and at any instant is often substantially lower than the expected maximum load [17, 19, 24, 26, 32]. Second, for most modern packet processing ap-

[1]This amount does not even include the capital investment required for installing cooling systems and Uninterruptible Power Supplies (UPS) necessary for operating these systems, nor the operating expense for the energy consumed by the cooling system.

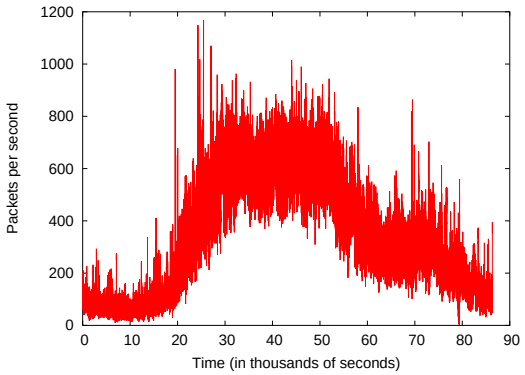

Figure 2: Packet arrivals per second over a day for the Auckland trace [23]

plications, the time to process a packet is dominated by memory-access latencies. Hence, an architecture containing a single, high-performance processor is often not suitable for a PPS. To mask memory access latencies, and thereby process packets at high rates, most modern PPSs utilize multiple parallel processors. For instance, Intel's IXP2800 network processor—a building block used in a wide-range of PPSs—includes 16 RISC cores (referred to as *microengines*) and an XScale controller.

The existence of multiple processors in PPSs when coupled with the fact that these systems often run at low levels of average utilization indicates that significant number of the available processors within a system can often be deactivated or run in a low-power mode. Together, these factors offer a significant opportunity to conserve energy.

The Challenges Although the properties of network workloads and of packet processing hardware raise opportunities for energy management, they also raise key challenges. First, because network traffic can fluctuate at multiple time-scales, accurately predicting traffic arrival patterns is difficult [19, 24, 26, 32] and intervals of idleness or low load may often be short [17], so it may be difficult to take advantage of periods of low load. Second, transitioning a processor from its activated to its deactivated state (and vice versa) often (temporarily) increases its rate of power consumption compared to leaving it activated but idle. Hence, state transitions should be performed only if a processor will remain deactivated long enough to conserve energy despite this transition cost. Third, switching a processor between its activated state and its deactivated state incurs a delay (generally of the order of a few hundred microseconds [18]). If the system deactivates processors too aggressively during idle periods, then because of the inherent delay in transitioning processors to their activated state, a burst of arriving packets might suffer unacceptable delays or losses.

Our Contributions In this paper, we present an on-line algorithm for adapting the number of activated processors at run-time. The algorithm minimizes the number of processors that are activated simultaneously and thereby provably minimizes energy consumption, and it assumes a worst-case packet arrival rate in order to deterministically bound the additional delay a packet can suffer as a result of the system's power management adaptations.

The key ideas in our PMA (Power Management Algorithm) and CPMA (Conservative Power Management Algorithm) are simple. In particular, like active queue management algorithms [2, 6, 11], our power management algorithms make decisions based only on the current queue length and do not need to predict future arrival patterns beyond knowing the worst-case arrival rate.

- To increase the number of activated processors, given a worst-case acceptable delay D, the PMA algorithm activates the $k+1$st processor when k processors are unable to process within D the sum of (a) all currently enqueued incoming packets and (b) the maximum number of packets that could arrive during a processor's activation latency. Surprisingly, for realistic system configurations, a simple sufficient condition to meet this general activation requirement is to activate the $k+1$st processor when the set of enqueued packets first exceeds the number of packets that k processors can process within D.
- Conversely, when the system has excess capacity, PMA deactivates one or more processors when both (a) the input queue becomes empty and (b) the minimum time until the processors would be reactivated under a worst-case arrival rate times the processor's power consumption in the activated state when idle exceeds the energy consumed by transitioning from activated to deactivated and back.

Whereas PMA aggressively exploits the processing delay bound to minimize power, the CPMA variation is slightly more conservative about deactivating processors and thus reduces the average packet delay imposed by power management at a modest cost to energy efficiency.

Although PMA and CPMA are simple, they have three significant advantages over existing systems. First, we show that the algorithms minimize the total energy consumption of the system over a packet trace. In particular, we show that for acceptable values of the delay bound D, PMA consumes the same amount of energy as an off-line optimal algorithm that utilizes future knowledge of packet arrival rates. Second, neither algorithm requires prediction of future packet arrival patterns. Given the variability of packet arrival rates in many network environments [19, 24, 26, 32], algorithms that do not depend on on-line prediction are likely to be less complex and more effective than those that do. Third, both algorithms deterministically meet a configurable bound on packet processing delay. This property stems directly from their

use of worst-case arrival rates rather than predicted arrival rates. Using trace-based workloads, we demonstrate that the algorithms work well in practice, reducing the core-processing power consumption by 60-70% while increasing the average packet processing delay for the different traces by between 1μs and 550μs (PMA) and between 1μs and 170μs (CPMA).

The rest of the paper is organized as follows. In Section 2, we describe our system model and formulate the problem of energy-efficient packet processing. In Sections 3 and 4, we discuss our power management algorithms. We describe the results of our experimental evaluation in Section 5. Related work is discussed in Section 6, and finally, Section 7 summarizes our contributions.

2 System Model

To develop our energy conservation algorithm, we consider a packet processing system (PPS) with N_p processors. Each packet arriving into the system is stored in a queue until a processor becomes available to service the packet (see Figure 3). The arrival rate of packets into the queue may fluctuate over time; the maximum rate of packet arrivals into the queue is given by R_{arr}. We assume that a packet can be served by any of the processors; servicing a packet takes t_{pkt} time units; and packets are serviced in the order of their arrival.

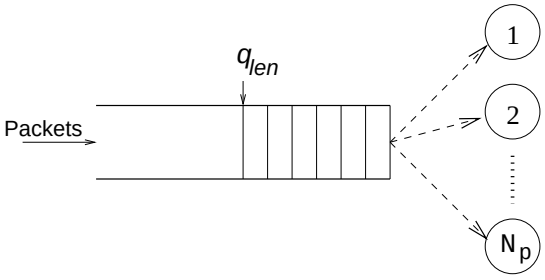

Figure 3: System model

Each processor in a PPS can be in either *activated* or *deactivated* state. In the deactivated state, we assume that the processor consumes no power.[2] [3] In the activated state, the processor can be busy processing packets or be idle; P_{active} and P_{idle}, respectively, denote the power consumption levels of a processor when it is busy and idle.

[2]Throughout this paper, we assume that in the the deactivated state, processors are turned off and hence consume no power. The deactivated processor state can also represent a *deep-sleep* state—that consume a small amount of power—supported in many modern processors; it is easy to generalize our model to such processors.

[3]Note that although for simplicity this paper focuses on a hardware model where power adaptation involves transitioning between activated and deactivated states, the PMA and CPMA algorithms are instantiations of a more general family of algorithms that can exploit more sophisticated hardware that provides frequency and voltage scaling configurations. We describe this generalization in an extended technical report.

Parameter	Description
N_p	number of processors in the system
R_{arr}	Worst-case arrival rate
t_{pkt}	Processing time per packet
P_{active}	Active power (activated state)
P_{idle}	Idle power (activated state)
t_{sw}	Switching delay (activated $\leftrightarrow$ deactivated)
P_{sw}	Switching power (activated $\leftrightarrow$ deactivated)
D	Delay guarantee on packet processing

Table 1: System model parameters

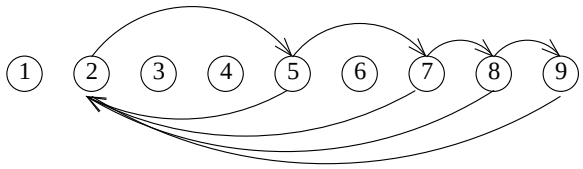

Figure 4: The finite-state automaton for a PMA system. The numbers in each state denote the number of activated processors.

We assume that transitioning the processor from the activated to the deactivated state or vice versa incurs a delay t_{sw} and consumes power P_{sw}. Although reducing the number of activated processors can save power, it may increase the queuing delay suffered by packets. Our algorithms bound the maximum delay D between when a packet is enqueued and when its processing is complete. Table 1 summarizes these system model parameters.

Observe that throughout our analysis, we only model a single *processing queue*. A PPS may have multiple input and output ports. Packets arriving on any of the input ports are queued for processing; upon being serviced by a processors, packets are queued for transmission at an appropriate output queue. We do not model delay incurred by packets in the input or output ports; we only model the delay incurred by packets while waiting for service by one of the processors.

3 Power Management Problem

Consider a packet processing system that is provisioned with a sufficient number of processors to meet the demands of the expected maximum load. To conserve energy, an adaptive system should maintain only as many activated processors as needed to process arriving packets before their processing deadlines.

Our power management algorithm (PMA) represents a packet processing system as a *finite state automaton (FSA)* (see Figure 4). Each state in the FSA represents the number of activated processors in the system. Transi-

tions from one state to another are triggered based on the length of the queue q_{len}.

When the system is in state j (i.e., with j activated processors), the system makes a transition to state $(j + k_a^j)$ (by activating an additional k_a^j processors) when the queue length exceeds a threshold Q_{on}^j. Similarly, when the queue is empty ($q_{len} = 0$), then from any state j, the system makes a transition to state n_{min} by deactivating $(j - n_{min})$ processors. In what follows, we describe construction of the FSA by deriving the values of Q_{on}^j and k_a^j for all values of j, and the value of n_{min}.

3.1 Calculating Q_{on}^j: When to Activate Processors?

We activate one or more processors when the queue grows to a level such that the delay incurred by packets can exceed the desired bound. The rate at which packets are serviced from the queue is a function of the number of activated processors. In particular, since each processor can service a packet in t_{pkt} time, the service rate R_{dep}^j of a collection of j activated processors is given by:

$$R_{dep}^j = \frac{j}{t_{pkt}} \quad (1)$$

Let Q_{lim}^j denote the maximum queue length that j processors can consume within the maximum permitted packet-processing delay D. Note that if there are Q_{lim}^j packets in the queue and the system has j activated processors, then the total number of packets in the system is $Q_{lim}^j + j$, so Q_{lim}^j is determined by

$$\begin{aligned} Q_{lim}^j + j &= D \times R_{dep}^j \\ \Rightarrow Q_{lim}^j &= j * \left(\frac{D}{t_{pkt}} - 1 \right) \end{aligned} \quad (2)$$

Suppose the arrival of packet p causes the queue length to become $Q_{lim}^j + 1$ (and hence the total number of packets in the system to become $Q_{lim}^j + 1 + j$). Then, with only j activated processors, the delay incurred by packet p would exceed D. To ensure that packet p can be serviced prior to its delay bound, one or more additional processors must be activated. Let us assume that the system initiates the activation of an additional processor τ units of time *prior* to the arrival of this packet p (see Figure 5).

Observe that the newly activated processor will become available for serving packets only after t_{sw} time units. Hence, for an interval $(t_{sw} - \tau)$ after the arrival of packet p, packets are serviced with j activated processors (and hence, at the rate of j/t_{pkt}); after that time, $(j+1)$ processors service packets at the rate of $(j+1)/t_{pkt}$. But, the delay bound D for packet p will be met only if packet p

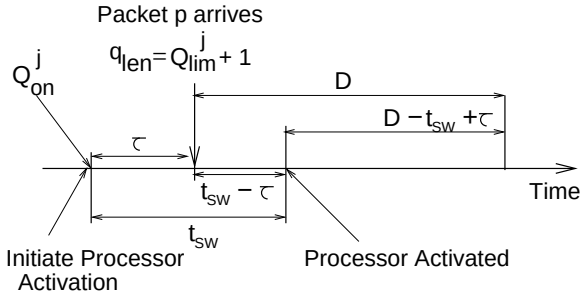

Figure 5: Activation procedure: Timing diagram

and all packets in front of p can be serviced within D units of time from the arrival of packet p. In particular, packet p will meet its delay bound D if and only if

$$Q_{lim}^j + 1 + j \quad \leq \quad (t_{sw} - \tau) \times \frac{j}{t_{pkt}} + (D - t_{sw} + \tau) \times \frac{j+1}{t_{pkt}}$$

This constraint requires

$$\tau \quad \geq \quad t_{pkt} + t_{sw} - D \quad (3)$$

which lets us determine a general equation for Q_{on}^j:

$$Q_{on}^j = Q_{lim}^j - \tau(R_{arr} - R_{dep}^j) \quad (4)$$

The above two expressions lead to the following important result:

Conclusion 1 *For $D \geq t_{sw} + t_{pkt}$, $\tau \leq 0$ and Q_{on}^j may simply be set to $Q_{on}^j = Q_{lim}^j$, indicating that the system can be completely reactive, adapting the number of activated processors only when the queue length $q_{len} > Q_{lim}^j$, where j is the number of currently activated processors.*

Conclusion 1 indicates that if the delay bound is greater than $(t_{pkt} + t_{sw})$, then the system is required to activate additional processors only *after* receiving a packet p whose delay bound would be violated with the current set of activated processors. In such systems, PMA therefore sets

$$Q_{on}^j = Q_{lim}^j$$

This result is important; it completely eliminates from most power management systems any need for predicting overload. Thus, when the delay bound $D > t_{pkt} + t_{sw}$, the system simply observes the queue build up and reacts

only when it receives a packet whose delay guarantee will be violated.[4]

The above condition is likely to be met by most realistic system configurations. For any system, the delay bound D must be at least t_{pkt} (the time required to process a packet). And given typical configuration switching delays of a few hundred microseconds and given that increasing a system's processing delay bound D increases a system's opportunities for power savings, we would expect most power-sensitive systems to operate with $D \gg t_{sw} + t_{pkt}$. Note, however that Equations 3 and 4 remain valid even if $D < t_{sw} + t_{pkt}$. In that case, $Q^j_{on} < Q^j_{lim}$ and PMA still begins activating processor $j+1$ sufficiently early to accomodate the worst-case packet arrival rate R_{arr}. Also note that in that case, the values of Q^j_{on} can still be calculated statically based on t_{pkt}, t_{sw}, D, and R_{arr}; no run-time prediction of arrival rates is needed.

3.2 Calculating k^j_a: How Many Processors to Activate?

Once the transition from deactivated to activated state is initiated for a processor, it becomes available to service packets only after a delay of t_{sw} time units. Thus, the number of processors to be activated is selected such that all the packets that can arrive within time t_{sw} (not just packet p that triggered the activation) can be serviced prior to their respective deadlines (defined by the delay bound).

If j and k^j_a, respectively, denote the number of currently activated processors and the ones being transitioned to the activated state, then the above condition can be met if the queue length at time when $(j + k^j_a)$ processors are activated does not exceed $Q^{j+k^j_a}_{on} + 1$. Note that at the time of initiating processor activations, the queue length was $Q^j_{on} + 1$. During the transition interval of length t_{sw}, packets can arrive into the system at a rate no greater than R_{arr}; further, with j activated processors, packet depart the system at rate R^j_{dep}. Thus, the maximum increase in queue length is bounded by $(R_{arr} - R^j_{dep}) \times t_{sw}$. Thus, the delay bound for each packet can be satisfied if:

$$(Q^{j+k^j_a}_{on} + 1) - (Q^j_{on} + 1) \quad \geq \quad \left(R_{arr} - R^j_{dep}\right) \times t_{sw}$$

Substituting the values for $Q^{j+k^j_a}_{on}$, Q^j_{on}, and R^j_{dep} from (1) and (4), we get:

$$k^j_a \times \left(\frac{D}{t_{pkt}} - 1\right) + \tau \times \left(\frac{k^j_a}{t_{pkt}}\right) \;\geq\; \left(R_{arr} - \frac{j}{t_{pkt}}\right) \times t_{sw}$$

$$\Rightarrow \quad k^j_a \;\geq\; \frac{(R_{arr} \times t_{pkt} - j) \times t_{sw}}{D + \tau - t_{pkt}} \quad (5)$$

This leads to the following conclusion.

Conclusion 2 *If the current number of activated processors is j and the queue length reaches Q^j_{on}, then the lower bound on the number of processors k^j_a that must be activated is given by:*

$$k^j_a = \left\lceil \frac{(R_{arr} \times t_{pkt} - j) \times t_{sw}}{D + \tau - t_{pkt}} \right\rceil \quad (6)$$

We make the following two observations.

- The value of k^j_a shown in Conclusion 2 is a function of j ($j \in [0, N_p]$), the number of activated processors prior to initiating the activation of new processors. The smaller the value of j, the greater is the value of k^j_a, and vice versa. This relationship allows the system to ramp-up quickly from a low-utilization state (with very few activated processors) by activating a larger number of processors with each activation trigger. The number of processors activated with each trigger decreases at higher levels of utilization.
- Packet processing systems are often provisioned to meet the demands of the expected maximum arrival rate R_{arr}. In such an appropriately provisioned system, the number of processors N_p is, in fact, equal to $R_{arr} \times t_{pkt}$. Thus, when $j = N_p$ (i.e., when all the processors in the system are in the activated state), $k^j_a = 0$. Further, in such a system, if $D > N_p \times t_{sw} + t_{pkt} - \tau$, then for all values of j, $k^j_a = 1$, i.e., each activation trigger will require the activation of exactly one additional processor.

3.3 Processor Deactivation

When processing capacity exceeds demand, processors should be deactivated to conserve energy.

3.3.1 When to Deactivate Processors?

When the queue of packets waiting to be serviced by a processor is not empty, then all of the activated processors are busy (and hence are expending P_{active} power while serving packets). It is important to note that if the queue contains n packets, then the total amount of energy expended to process the packets is given by: $n \times t_{pkt} \times P_{active}$, which is independent on the number of activated processors serving the packets. Thus, deactivating a processor while the queue is not empty does not conserve energy. Furthermore, deactivating processors before the queue is empty increases the waiting time for the packets. These observations lead us to the following conclusion.

Conclusion 3 *Processors should be deactivated only when the queue is empty.*

[4] In fact, if $D > (t_{pkt} + t_{sw})$, then τ can be negative, indicating that the system can even wait for $|\tau|$ time units after the arrival of packet p before activating a new processor. Note, however, that such additional delay does not reduce the total amount of energy expended by the system for a trace because it does not reduce the amount of time that processor $j+1$ must work to ensure the system meets its packet delay bounds.

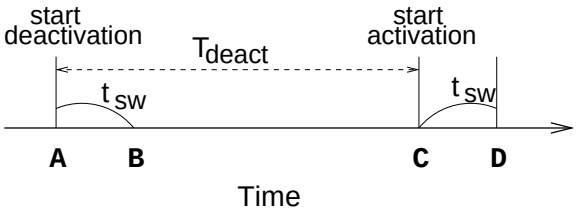

Figure 6: Condition to Save Power

3.3.2 How Many Processors to Deactivate?

Once the queue is empty, processors can be deactivated to conserve energy. To determine the number of processors that can be deactivated, we first derive the minimum number of processors that must remain activated beyond which deactivating processors further may not yield any reduction in energy consumption.

Let n_{min} be the minimum number of processors that must remain activated in order to ensure that deactivating the remaining processors is profitable. To derive the value of n_{min}, we observe that deactivating a processor is beneficial only if doing so conserves energy. In particular, we will guarantee that if n_{min} processors remain active, then the $(n_{min}+1)$st processor will remain deactivated long enough that its cost of transitioning to and from the deactivated state (at an energy cost of $2.P_{sw}.t_{sw}$ is less than the cost of remaining activated and idle, consuming P_{idle} for that time.

Observe that any more processors than n_{min} would need to be activated only τ units of time prior to the instant when the queue length reaches $Q_{lim}^{n_{min}}+1$ (from its current value of 0). Given that packets can arrive at the maximum rate R_{arr} and that packets are serviced at rate $R_{dep}^{n_{min}}$ with n_{min} activated processors, the minimum time processors (other than the always-activated n_{min} processors) remain deactivated is given by:

$$T_{deact} = \frac{Q_{lim}^{n_{min}}+1}{R_{arr}-R_{dep}^{n_{min}}}-\tau \quad (7)$$

Let j be the current number of activated processors. Consider the sequence of events shown in Figure 6. The total amount of energy expended in transitioning $(j-n_{min})$ processors from activated state to deactivated state and back is given by: $2\times P_{sw}\times t_{sw}\times(j-n_{min})$. In comparison, by not deactivating $(j-n_{min})$ processors, the system would have expended $(t_{deact}+t_{sw})\times P_{idle}\times(j-n_{min})$ energy, given that activated processor only expend P_{idle} power while being idle. Thus, deactivating $(j-n_{min})$ processors conserves energy only if:

$$2\times P_{sw}\times t_{sw}\times(j-n_{min}) \leq (T_{deact}+t_{sw})\times P_{idle}\times(j-n_{min})$$

$$\Rightarrow T_{deact}\geq\left(2\times\frac{P_{sw}}{P_{idle}}-1\right)\times t_{sw} \quad (8)$$

Using Equations 7 and 8, we get:

$$\frac{Q_{lim}^{n_{min}}+1}{R_{arr}-R_{dep}^{n_{min}}}-\tau \geq \left(2\times\frac{P_{sw}}{P_{idle}}-1\right)\times t_{sw} \quad (9)$$

$$(10)$$

which gives us an equation for n_{min} that is independent of j

$$\forall j: n_{min} \geq \frac{(C\times R_{arr}-1)\times t_{pkt}}{D-t_{pkt}+C} \quad (11)$$

$$where\ \ C=\left(2\times\frac{P_{sw}}{P_{idle}}-1\right)\times t_{sw}+\tau$$

Using Equation 11, we derive the following conclusion.

Conclusion 4 *Once the queue becomes empty, to conserve energy, an adaptive system should deactivate all but n_{min} processors.*

Observe from Equation 9 that if

$$\tau\leq-\left(2\times\frac{P_{sw}}{P_{idle}}-1\right)\times t_{sw} \quad (12)$$

then Equation 9 can be satisfied for all values of n_{min}, including $n_{min}=0$. Substituting this value of τ in Equation 3, we get:

$$D\geq t_{pkt}+t_{sw}-\left(-\left(2\times\frac{P_{sw}}{P_{idle}}-1\right)\times t_{sw}\right)$$

$$\Rightarrow D\geq t_{pkt}+2\times\left(\frac{P_{sw}}{P_{idle}}\right)\times t_{sw} \quad (13)$$

Thus, if D satisfies Equation 13, then $n_{min}=0$. We summarize this in the following conclusion.

Conclusion 5 *If D satisfies Equation 13, then by selecting τ to satisfy Equation 12, one can design an adaptive system in which once the queue becomes empty, the system can deactivate all the idle processors. This maximizes the conserved energy, while ensuring that the delay incurred by packets is within bound D.*

3.4 Analysis

In this section, we compare our power management algorithm (PMA) to an optimal, off-line algorithm (referred to as ORAC). By virtue of being an off-line algorithm, at any time instant, ORAC adapts processor activations based on exactly the observed traffic in the future. Thus, ORAC activates processors only when needed; further, it

activates only as many processors as needed. Similarly, when the packet trace contains sufficiently long idle durations, ORAC deactivates all of the processors. Now, let us consider PMA.

- As we have argued in Section 3.1, if $D \geq t_{pkt} + t_{sw}$, then $\tau \leq 0$. Thus PMA is completely reactive; it activates a processor only after receiving a packet whose delay guarantee can't be honored with the current set of activated processors. Hence, when $D \geq t_{pkt} + t_{sw}$, PMA will activate processors only when ORAC also activates processors.

- On receiving an activation trigger in state j (with j activated processors), however, PMA activates an additional k_a^j processors. The derivation of k_a^j assumes that the traffic could arrive at its maximum rate R_{arr}. Thus, in comparison with ORAC, PMA could activate at most $(k_a^j - 1)$ additional processors, and thereby incur $(k_a^j - 1) \times t_{sw} \times P_{sw}$ energy overhead for each activation trigger.

  However, as we have argued in Section 3.2, if $D > N_p \times t_{sw} + t_{pkt} - \tau$, then for all values of j, $k_a^j = 1$. In this case, PMA will activate exactly one processor for each activation trigger, and hence will match the energy consumption of ORAC.

- Both PMA and ORAC deactivate processors only when the queue becomes empty. However, unlike ORAC, PMA deactivates all but n_{min} processors. PMA maintains n_{min} activated processors because of the assumption that a burst of packets can arrive at any instant at the maximum arrival rate R_{arr}; hence, deactivating them would not conserve any energy. If, in fact, the packet trace contains long idle spans, then ORAC would deactivate all of the processors. For each such idle span, the energy consumed by PMA is higher than ORAC by $n_{min} \times P_{idle} \times I$, where I denotes the length of the idle span.

  However, as we have demonstrated in Section 3.3, if τ is selected such that $\tau \leq -\left(2 \times \frac{P_{sw}}{P_{idle}} - 1\right) \times t_{sw}$, and if $D \geq t_{pkt} + 2 \times \left(\frac{P_{sw}}{P_{idle}}\right) \times t_{sw}$, then $n_{min} = 0$. In such a case, once the queue becomes empty, PMA would deactivate all the idle processors, and thereby match the energy consumption of ORAC.

From the above discussion, we derive the following conclusion.

Conclusion 6 *If the delay bound D satisfies:*

$$D \quad \geq \quad N_p \times t_{sw} + t_{pkt} + \left(2 \times \frac{P_{sw}}{P_{idle}} - 1\right) \times t_{sw}$$

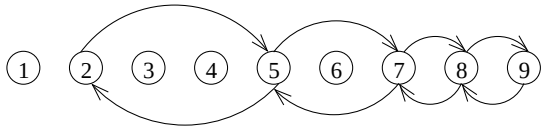

Figure 7: CPMA Automata

then our on-line power management algorithm consumes the same amount of energy as an optimal off-line algorithm.

4 Conservative PMA

In our power management algorithm (PMA) described in Section 3, when the queue is empty, PMA deactivates all but n_{min} processors. This policy aggressively exploits the processing delay bound to minimize energy consumption. This increases the average as well as the variance in packet delay. In this section, we propose a variant of PMA (which we refer to as *Conservative PMA*) that is slightly more conservative about deactivating processors and thus reduces the average packet delay at a modest cost to energy efficiency.

CPMA estimates the actual departure rate of packets from the system as:

$$\widehat{R}_{dep} \quad = \quad \frac{n_{pkt}}{\Delta} \tag{14}$$

where n_{pkt} denotes the number of packets departed in time Δ, and Δ is the time since the most recent processor activation. If j is current the number of activated processors in the system, then the maximum rate for serving packets is given by $R_{dep}^j = j/t_{pkt}$, where t_{pkt} is the time to service a single packet. Let k be the number of activated processors in the state previous to j in the FSA.

CPMA then evaluates the following condition:

$$\begin{aligned} \widehat{R}_{dep} \quad &\leq \quad R_{dep}^k \\ \Rightarrow \quad \frac{\widehat{R}_{dep}}{R_{dep}^j} \quad &\leq \quad \frac{k}{j} \end{aligned} \tag{15}$$

When the queue is empty, only if Equation 15 holds, CPMA deactivates processors. Further, unlike PMA (which would deactivate all but n_{min} processors), CPMA deactivates only $(j-k)$ processors. The FSA resulting from CPMA is shown in Figure 7.

5 Experimental Evaluation

We evaluate the energy benefits of PMA and CPMA over a base system B – a non-adaptive system that keeps all processors always activated. For reference, we also compare with (1) ORAC — a system that uses perfect knowledge of the future to predict processor allocations, and (2) MAX — the maximum possible benefits when there are no switching overheads.

Metric→ ↓Trace	Duration	Size (in Million)	N_p	Min on Processors
UNC1	7 mins	10	27	5
UNC2	5 mins	10	27	5
UNC3	16 mins	30	27	5
UNC4	8 mins	30	17	5
NZIX1	6 hrs	23.4	6	2
NZIX2	6 hrs	26.9	4	2
Bell1	24 hrs	47.8	12	3
Bell2	24 hrs	51	12	3
AUCK	24 hrs	28.7	9	3

Table 2: Trace characteristics. T_{min} is the minimum inter-arrival time observed in the trace. T_{pkt} is calculated assuming that $max_{proc} = 16$.

Parameter	Value
D	2 ms
t_{sw}	200 μs
t_{pkt}	200 μs
$\frac{P_{sw}}{P_{idle}}$	1.3
$\frac{P_{active}}{P_{idle}}$	1.3

Table 3: System Parameter Values for experiments.

5.1 Setup

Metric: We derive the metric of evaluation as follows: For any of the algorithms (PMA, CPMA, ORAC and MAX) represented by A, we define the energy benefits over the base system B as $1 - \frac{E_A}{E_B}$, where $E_A = P_{active} \times T^A_{active} + P_{idle} \times T^A_{idle} + P_{sw} \times t_{sw} \times Trans^A$, and $E_B = P_{active} \times T^B_{active} + P_{idle} \times T^B_{idle}$. T^A_{state} represents the total time the system spent in a particular state (idle, active or off) for a particular algorithm. *Trans* represents the number of transitions between activated and deactivated states.

Traces: We use both synthetic and real world traces to evaluate the energy benefits of our algorithms. We construct a synthetic STEP trace to compare the behavior of PMA and CPMA. To construct the STEP trace, we first increase the number of packets in each uniform sized interval and then decrease it symmetrically, forming a series of rising and falling steps. We use several real-world traces [23, 27] of various durations and varying density of traffic, collected from various Internet locations. Columns 2 and 3 of Table 2 show the characteristics of these traces.

System Parameters: Table 3 shows the parameters we pick for our experiments. We derive the value for $\frac{P_{active}}{P_{idle}}$ and $\frac{P_{sw}}{P_{idle}}$ as 1.3, using the ratio of active to idle power for IXP 1200 from Figure 1. We pick $t_{sw} = 200\mu$s [18] and choose $t_{pkt} = 200\mu$s extrapolating data from [17] to Intel IXP 1200.

Trace	PMA	CPMA	ORAC	MAX
UNC1	73.60	73.32	77.30	77.68
UNC2	69.72	62.39	71.68	72.04
UNC3	69.65	62.78	71.62	71.98
UNC4	60.13	60.06	69.07	69.62
NZIX1	65.95	65.95	90.58	95.35
NZIX2	49.08	49.08	84.30	92.04
BELL1	66.48	66.48	97.30	98.80
BELL2	66.47	66.47	97.11	98.72
AUCK	55.43	55.43	97.73	99.04

Table 4: Comparison of percentage benefits obtained by PMA with ORAC and MAX.

Provisioning: In order to make a fair comparison, we allow the base system to keep the same size of the queue that our algorithms need. We provision the system with enough number of processors such that the maximum number of packets received in any interval D (the delay guarantee) throughout the trace, can be kept in the queue and processed within D. Note that this is a conservative assumption – in reality, systems are typically over-provisioned to handle the peak traffic arrival rates. We use $t_{pkt} = 200\mu$s for deriving the maximum processor requirement. Column 4 shows the processor provisioning required for each trace. Note that once we know all these values, we can derive the minimum number of processors (column 5) that are always kept on from Equation 11.

5.2 Results

In this section, we demonstrate the efficacy of our algorithms. Throughout this section, we assume that $\tau = 0$. As we have argued earlier, selecting $\tau = 0$ reduces the energy-efficiency of PMA as compared to ORAC; using $\tau < 0$ would improve the observed benefits further. Our choice of τ allows us to study the effect of varying delay guarantee on energy benefits, and ensures simplicity of the presentation.

Comparison to ORAC: Table 4 compares the benefits achieved by PMA and CPMA to ORAC for different traces. The table shows that PMA and CPMA achieve substantial benefits as compared to ORAC. Recall that while PMA keeps certain number of processors always switched on, ORAC can turn off all processors in the system. This explains the difference in benefits for the longer duration NZIX and BELL traces — these traces require no processors for significant periods of time.

Figure 8 shows the delay CDF of packets for different traces using PMA. The figure shows that although the maximum allowed delay is 2ms, most packets see significantly smaller delays. In the case of UNC, the delays are

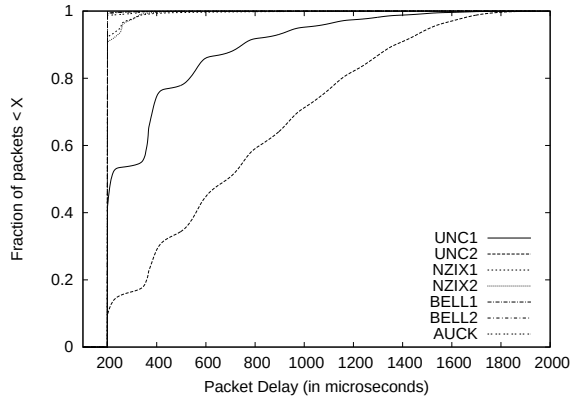

Figure 8: CDF of the delays seen by packets when using the two algorithms.

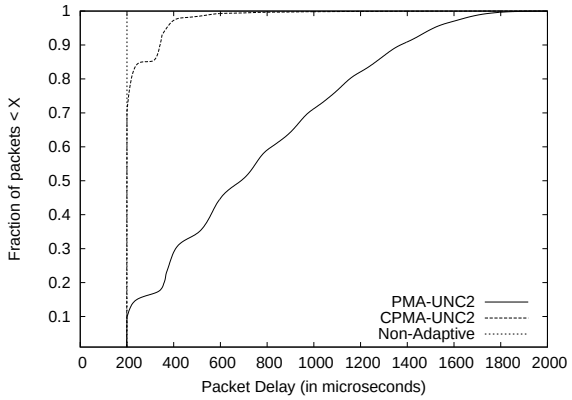

Figure 10: CDF of the delays seen by packets when using the two algorithms.

longer because the UNC traces have higher packet density that the others. The high packet density combined with the aggressive deactivation of PMA (when the queue reaches zero), causes the queue to build up frequently, thus causing long packet delays.

Comparison between PMA and CPMA: Figure 9 highlights the difference between PMA and CPMA in terms of the aggressiveness in saving energy. Figure 9(a) shows the behavior of PMA (in terms of the queue length and the number of processors activated) with time for the STEP trace. The fluctuations in the graph illustrate the aggressiveness of PMA in deactivating processors every time the queue length reaches zero. Figure 9(b) shows the behavior of CPMA for the same trace. Recall that CPMA deactivates a processor only when the utilization is low enough that the processor is not necessary, hence reducing the number of transitions.

Metric →	Transitions per sec		Avg Extra Delay (μs)	
Trace ↓	PMA	CPMA	PMA	CPMA
UNC1	90	79	178.09	168.27
UNC2	1654	7	548.66	31.27
UNC3	1684	27.57	552.52	44.48
UNC4	7.19	5.06	73.49	71.22
NZIX1	0.042	0.035	7.997	7.83
NZIX2	0.00083	0.00064	7.038	7.034
BELL1	0.0152	0.015	0.934	0.952
BELL2	0.0097	0.0096	1.177	1.192
AUCK	0.0003	0.0003	1.356	1.356

Table 5: Comparison of Aggressive and Conservative algorithms in terms of transitions and average packet delay.

Table 5 compares the number of transitions and the average delay caused by PMA and CPMA for various traces. The table shows that CPMA is better in reducing the number of transitions and average delay. The performance is significantly better for the busier traces (UNC). Figure 10 plots the CDF of the delay experienced by packets with PMA and CPMA for the UNC2 trace and compares them with the non-adaptive system. The graph illustrates that while PMA delays 50% of the packets by more than 600μs, CPMA causes no delays to 70% of the packets.

Effect of Varying Delay Guarantee Figure 11 shows the effect of changing the delay guarantee on energy benefits and average packet delay for UNC1 trace using PMA. The graph shows that by increasing the delay guarantee we see increased benefits till $D = 2$ms. However, further increase in delay guarantee does not increase the benefits significantly. The average delay also increases with the increase in delay guarantee. Observe here that around $D = 1.5$ms, we get almost all the benefits, with almost no increase in average delay.

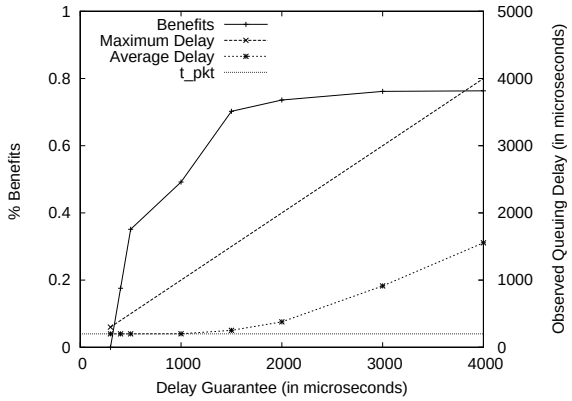

Figure 11: Effect of changing delay guarantee on the benefits for the UNC trace, average delay and maximum delay observed.

Effect of switching overheads We study the effect of varying switching delay (t_{sw}) and switching power (P_{sw}) on energy benefits. Figure 12 shows that as the switching delay and power increase, the benefits reduce and drop to

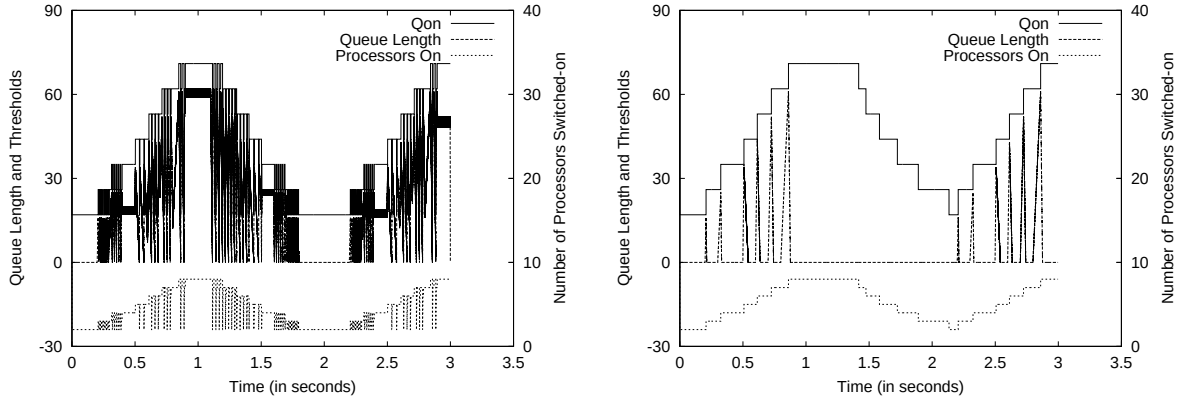

Figure 9: Comparison of PMA and CPMA in terms of variation of the threshold, the queue length and the number of activated processors for the step trace. The figure clearly depicts the conservative behavior of CPMA when compared to PMA.

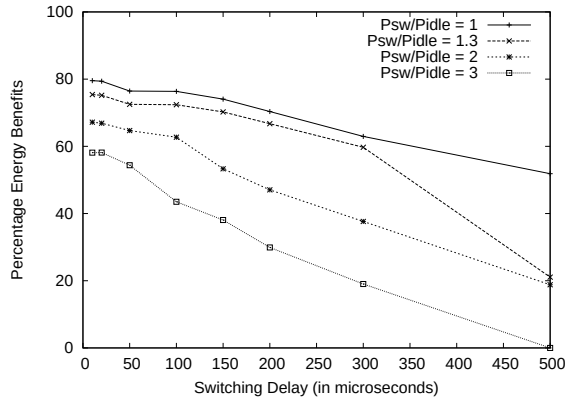

Figure 12: Variation of power benefits with switching delay and switching power for the UNC trace.

zero. However, for today's IXP processors [18], switching delay is around 200μs and from Figure 1, switching to idle power ratio is about 1.3. From Figure 12, these values correspond to energy benefits of 70%. In future, the switching overheads are only likely to reduce, given the increasing focus on energy-aware system designs.

6 Related Work

Several recent studies have shown the benefits and opportunities of conserving energy in packet processing systems, and argued for building energy-aware networking devices and protocols [1, 5, 14, 15, 17]. Gupta et al. [14] motivate that conserving energy reduces the operational costs of packet processing systems, reduces the environmental impact (e.g. heat dissipation and cooling-induced noise) and also leads to the greater deployment of the Internet. Bhagwat et al. [5] point out the need for energy-efficient devices for their real-life networking testbed – which operates on solar generated power. Kokku et al. [17] show that the inherent fluctuations in network traffic leave significant opportunity for an adaptive environment to adapt processor allocations for conserving energy. Power efficiency in network processors has been a topic of growing interest [12, 21, 22, 28].

The problem of scheduling resources for conserving energy has been explored in various domains – wireless and sensor networks [8, 13, 16, 31], mobile and embedded systems [9, 25, 29], web servers [7, 10], and multimedia servers [30]. Several recent works develop protocols and policies for adaptive resource management to conserve energy [4]. However, in packet processing systems like routers, hubs and switches, the problem has received little focus. Our work is motivated by the recent realization of the need for system support for energy-conservation in such systems [1, 5, 14, 15].

Managing energy in packet processing systems becomes challenging due to two factors (1) bursty network traffic and lack of predictability, and (2) non-negligible cost of adapting network resources at run-time. Over several years, network traffic has been shown to be bursty at various timescales and is especially hard to predict at fine timescales [19, 24, 26, 32].

Dynamic power management in several systems is done using predictive schemes and stochastic schemes. Predictive schemes use observation based-approaches like exponential averaging [4, 20] of task arrival rate and predict the resource requirements. With network traffic, such schemes could be inaccurate and thereby either waste resources or affect the performance guarantees with no bounds. Stochastic schemes formulate energy-conservation as a constrained optimization problem – they minimize power consumption under the performance constraints [3, 4]. However, solving an optimization problem at run-time may cause significant overhead.

Our algorithm uses Active Queue Management (AQM) to determine processor requirements under bounded per-

formance impact. AQM [2, 6, 11] has been a popular technique for congestion control in the Internet. The AQM abstraction fits well for packet processing systems, and thus makes the algorithm simple and low overhead to implement.

7 Conclusion

In this paper, we present an on-line algorithm for adapting the number of activated processors such that (1) the total energy consumption of the system across a packet trace is minimized and (2) the additional delay suffered by packets as a result of adaptation is deterministically bounded. The resulting Power Management Algorithm (PMA) is simple, but it accounts for system reconfiguration overheads, copes with the unpredictability of packet arrival patterns, and consumes nearly the same energy as an optimal off-line strategy. A conservative version of the algorithm (CPMA), turns off processors less aggressively than is optimal but still provides good energy savings while reducing the additional packet latency introduced by power management. We demonstrate that for a set of trace workloads both algorithms can reduce the core power consumption by 60-70% while increasing the average packet processing delay by less than 560μs (PMA) and less than 170μs (CPMA).

References

[1] *International Energy Agency Workshop* on The Future Impact of Information and Communication Technologies on the Energy System, Paris, February 21-22, 2002. `www.worldenergyoutlook.org/weo/papers/ictfeb02.asp`.

[2] S. Athuraliya, V. H. Li, S. H. Low, and Q. Yin. REM: Active queue management. *IEEE Network*, 15(3):48 – 53, May/June 2001.

[3] S. Baruah and J. Anderson. Energy-aware implementation of hard-real-time systems upon multiprocessor platforms. In *Proceedings of the ICSA 16th International Conference on Parallel and Distributed Computing Systems.*, pages 430–435, 2002.

[4] L. Benini, A. Bogliolo, and G. D. Micheli. A Survey of Design Techniques for System-Level Dynamic Power Management. *IEEE Transactions on Very Large Scale Systems*, 2000.

[5] P. Bhagwat, B. Raman, and D. Sanghi. Turning 802.11 Inside-Out. In *Hotnets II*, November 2003.

[6] W. chang Feng, K. G. Shin, D. D. Kandlur, and D. Saha. The BLUE active queue management algorithms. *IEEE/ACM Transactions on Networking*, 10(4):513–528, 2002.

[7] J. S. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing Energy and Server Resources in Hosting Centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP)*, October 2001.

[8] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. SPAN: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks. In *Mobile Computing and Networking*, pages 85–96, 2001.

[9] F. Douglis, P. Krishnan, and B. Bershad. Adaptive Disk Spindown Policies for Mobile Computers. In *Proceedings of the 2nd USENIX Symposium on Mobile and Location-Independent Computing*, 1995.

[10] M. Elnozahy, M. Kistler, and R. Rajamony. Energy Conservation Policies for Web Servers. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, March 2003.

[11] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[12] M. A. Franklin and T. Wolf. Power Considerations in Network Processor Design. In *Proceedings of Second Network Processor Workshop in conjunction with Ninth International Symposium on High Performance Computer Architecture (HPCA-9)*, pages 10–22, 2003.

[13] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly-Resilient, Energy-Efficient Multipath Routing in Wireless Sensor Networks. *Mobile Computing and Communications Review*, 4(5), 2001.

[14] M. Gupta and S. Singh. Greening of the Internet. In *Proceedings of the ACM SIGCOMM*, August 2003.

[15] G. Held. Emerging Technology: The Price of Power Consumption. Network Magazine, Sep 5, 2001.

[16] C. E. Jones, K. M. Sivalingam, P. Agrawal, and J.-C. Chen. A Survey of Energy Efficient Network Protocols for Wireless Networks. *Wireless Networks*, 7(4):343–358, 2001.

[17] R. Kokku, T. Riche, A. Kunze, J. Mudigonda, J. Jason, and H. Vin. A Case for Run-time Adaptation in Packet Processing Systems. In *Hotnets II*, November 2003.

[18] M. E. Kounavis, A. T. Campbell, S. T. Chou, and J. Vicente. Programming the Data Path in Network Processor-Based Routers. *Software Practice and Experience*, Special Issue on Software for Network Processors, 2004.

[19] W. Leland, M. Taqq, W. Willinger, and D. Wilson. On the self-similar nature of Ethernet traffic. In *Proceedings of the ACM SIGCOMM*, 1993.

[20] Y. Lu, E. Chung, T. Simunic, L. Benini, and G. Micheli. Quantitative Comparison of Power Management Algorithms. In *Proceedings of Design Automation and Test in Europe.*, 2000.

[21] G. Memik and W. H. Mangione-Smith. Increasing Power Efficiency of Multi-Core Network Processors through Data Filtering. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 108–116. ACM Press, 2002.

[22] G. Narlikar, A. Basu, and F. Zane. CoolCAMs: Power-Efficient TCAMs for Forwarding Engines. In *Proceedings of IEEE Infocom*, 2003.

[23] NLANR Network Traffic Packet Header Traces. http://pma.nlanr.net/Traces/.

[24] V. Paxson and S. Floyd. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.

[25] P. Pillai and K. G. Shin. Real-time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proceedings of the eighteenth ACM Symposium on Operating Systems Principles*, pages 89–102. ACM Press, 2001.

[26] Y. Qiao, J. Skicewicz, and P. Dinda. Multiscale Predictability of Network Traffic. Technical report.

[27] F. D. Smith, F. H. Campos, K. Jeffay, and D. Ott. What TCP/IP Protocol Headers Can Tell Us About the Web. In *Proceedings of ACM SIGMETRICS 2001/Performance 2001*, June 2001.

[28] E. Spitznagel, D. Taylor, and J. Turner. Packet Classification Using Extended TCAMs. In *Proceedings of ICNP*, 2003.

[29] M. Weiser, B. Welch, A. J. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation*, pages 13–23, 1994.

[30] Q. Wu, M. Pedram, and Q. Qiu. Dynamic Power Management in a Mobile Multimedia System with Guaranteed Quality-of-Service. In *Proceedings of the Design Automation Conference*, pages 701–707, 2001.

[31] W. Ye, J. Heidemann, and D. Estrin. An Energy-Efficient MAC protocol for Wireless Sensor Networks. In *Proceedings of the IEEE INFOCOM.*, 2002.

[32] Z. Zhang, V. Ribeiro, S. Moon, and C. Diot. Small-Time Scaling Behaviors of Internet Backbone Traffic: An Empirical Study. In *Proceedings of the IEEE INFOCOM.*, 2003.